CS494
RFC Document: IRC Class Project Specification
Joseph Jesse & Cintiha Rosas
Portland State University
June 1st, 2020

Abstract

This document describes and explains the communication protocol used by Joseph Jesse and Cintiha Rosas for an IRC-style client/server system for the Internetworking Protocols class at Portland State University taught by Niru Bulusu. To run the application, first run the server by invoking the main method in Server.java [hit run if using an IDE, preferably VS Code] and then invoke the main method in IPChat.java. In order to run our IRC, please make sure you have Oracle JDK (at least version 8) to run our GUI: Java SE - Downloads | Oracle Technology Network

1. Introduction

This record of specification describes a simple GUI based Internet Relay Chat (IRC) protocol from which clients can communicate with each other through a common server based on rooms created or joined. The system that we have implemented employs a central server which forwards messages and sends responses to client requests. Messages sent to a room are forwarded to all clients who are connected to the server and in the specified room. The users have the capability to join rooms, these rooms are used to create a stream of messages displayed for every user. Users are able to be in a room with multiple people, be in multiple rooms, and send distinct messages to each room.

2. Basic Information

All communication described in this protocol takes place with the server listening for connections on port number 7777. If the server is not able to connect to the port 7777 on its host machine, the process running on that port must be killed in order for the server to run. Once the server is running, clients are able to connect to this port and exchange packets with the server—unless the client or server were to crash unexpectedly. The client is able to send messages to a room and requests to the server through the user interface. Requests include listing all the rooms created in the application, joining a room, leaving a room, and displaying all the users in a room. Assuming no unusual circumstances or a crash, the server will in all cases respond to the client even in cases when an error has occurred. With this, messaging in the

application is inherently asynchronous; the client is able to send messages to the server at any time and the server will then concurrently forward them to the clients in the room as a response.

Just like the client can connect to the server at any point while it's running, the client and server can both choose to terminate connection at any time. The client is able to press the button on the GUI to send a terminating message to the server; the server will then respond confirming that it has received the notice of termination from the client. On the server side, it is able to disconnect from all current users in the system and notify them of its termination due to maintenance, deployments, etc. To do this, a system administrator is required to enter an admin password to the server process which will begin this process.

3. Message Infrastructure

3.1 Operation Codes (opcodes)

```
OP_CODE_ERR(0x00000001)
OP_CODE_KEEP_ALIVE(0x00000002)
OP_CODE_HELLO(0x00000003
OP_CODE_LIST_ROOMS(0x00000004)
OP_CODE_LIST_ROOMS_RESPONSE(0x00000005)
OP_CODE_LIST_USERS(0x00000006)
OP_CODE_LIST_USERS_RESP(0x00000007)
OP_CODE_JOIN_ROOM(0x00000008)
OP_CODE_JOIN_ROOM_RESP(0x00000009)
OP_CODE_LEAVE_ROOM(0x0000000A)
OP_CODE_LEAVE_ROOM_RESP(0x0000000B)
OP_CODE_SEND_MESSAGE(0x0000000C)
OP_CODE_SEND_MESSAGE_RESPONSE(0x0000000D)
OP_CODE_SEND_PRIVATE_MESSAGE(0x0000000E)
OP_CODE_TELL_PRIVATE_MESSAGE(0x0000000F)
OP_CODE_GOODBYE(0x00000010)
```

3.2 Error Codes

Specifies the type of error that occurred

```
IRC_ERR_UNKNOWN(0x00000001)
 IRC_ERR_ILLEGAL_OPCODE(0x00000002)
 IRC_ERR_ILLEGAL_LENGTH(0x00000003)
```

```
IRC_ERR_NAME_EXISTS(0x00000004)
IRC_ERR_ILLEGAL_PROTOCOL(0x00000005)
IRC_ERR_INVALID_ROOMNAME(0x00000006)
```

4. Protocol Specification

When the connection is first established between the client and server, the first thing that is exchanged is a hello/handshake packet. Of the many things this first exchange of data takes care of (see section 6.1 of this RFC document for more details) maybe the most important is the agreement of protocol exchange. This is specified explicitly with a code in the hello packet. The standards for the server application are set so that it is only allowed to communicate with clients using the protocol described by the hexadecimal code `0x12345678`. If any other protocol is specified by a client, there is an immediate error response sent by the server back to the client refusing the connection. The error packet is such that it contains an error opcode `OP_CODE_ERR(0x00000001)` and a specific error code `IRC_ERR_ILLEGAL_PROTOCOL(0x00000005)` informing the client of the protocol incompatibility.

5. Handling client crashes

5.1 Keepalive Messages

For many of the functions the server provides to clients—such as listing all users in a room, forwarding messages to all the users in a room, and notifying clients of server termination—the server needs to maintain information about the clients that are currently connected and running the application. Thus, to accurately maintain this information, and keep it updated, the server needs to periodically check which clients are still running the application. In the ideal case, clients will notify the server when they disconnect allowing the server process to remove them from the system. Unfortunately, this can not always be relied on. In some instances the client could crash leaving a state in which the client is no longer running the application and the server is unaware. Thus, a keep alive packet with the op code `OP_CODE_KEEP_ALIVE(0x00000002)` is sent in five second increments to every server tracked client. If a keep alive response is not returned from the client, the server assumes the client is no longer running the application and thus removes them from the system.

6. Client Messages/Requests

6.1 Client-server handshake

Prior to asynchronous communication between the server and the client, a connecting client must provide a username in order for a connecting. Following this, a “Handshake” packet with opcode `OP_CODE_HELLO(0x00000003)` is sent to the server with the username the client wishes to register as. When successful, the server stores the client’s username, the host, and the port number (for keep alive communications). One requirement of the client’s registered username is it must be unique in the system. If the client hands a non-unique name an error will be sent out to re-enter a unique username. This packet is to be sent to the server only when a client is first connecting.

6.2 Listing Rooms and Users

Upon loading of the GUI, the user is presented with a menu board. This menu board contains multiple action options that a user can perform—two of which being buttons titled “List all Rooms" and “List All Users in Room”. Listing all rooms is a request that sends a packet with opcode [`OP_CODE_LIST_ROOMS(0x00000004]` to the server. If the request is valid (proper opcode specified) the server returns a list rooms response packet containing the [`OP_CODE_LIST_ROOMS_RESPONSE(0x00000005)`] opcode and an ArrayList of Strings containing the chat room names currently occupied by at least one client. Listing all users in a room works very similarly except when pressing the menu button, the GUI creates a pop up window for the user to type the name of the room they wish to display the uses for. Using the list users packet [`OP_CODE_LIST_USERS(0x00000006)`] the request is sent to the server. Upon receiving it, the server returns the resulting response packet[`OP_CODE_LIST_ROOMS_RESPONSE(0x00000005)`] to the client which is then displayed. If there are no users in the room specified—implying the room name requested actually does not match that of one in the system, the room is displayed to be empty.

6.3 Joining and Creating Rooms

Another action the client can perform is to “Join a Room”. This action can be used for joining a pre-existing room or creating a brand new room—both are the same underlying request. The request is a packet[`OP_CODE_JOIN_ROOM(0x00000008)`] sent to the server with the Room name they wish to join/create. Contained in the packet is the username they gave upon

connecting to the server and their port number for communication with the server. The port number is used to relay information to the newly created client room thread. The server then responds with the packet [OP_CODE_JOIN_ROOM_RESP(0x00000009)] in all cases. There is no point of failure here being that room names are not restricted.

6.4 Leaving a Room

A client can request to leave any room it is currently subscribed to. The opcode for this request is [OP_CODE_LEAVE_ROOM(0x0000000A)] containing the name of the room to remove and the client's username. In this case however, error can occur. The first case is when the client requests to leave a room that does not exist in the system. This will result in an invalid room name response error packet from the server with error opcode IRC_ERR_INVALID_ROOMNAME(0x00000006). The second point of failure can be presented when the client names a valid room, yet they are not subscribed to that room. In this case, a valid response is sent back to the client instead of an error, yet nothing is done since there is nothing to do. If however the request is for a room the client is subscribed to, the server responds with a leave room response packet with opcode OP_CODE_LEAVE_ROOM_RESP(0x0000000B) indicating the client has been removed from the room.

6.5 Disconnecting From Server

To exit the application, the client sends a goodbye packet to the server containing the [OP_CODE_GOODBYE(0x00000010)] opcode. This notifies the server of the client termination. Assuming no errors in data exchange, the server will send a goodbye packet in response acknowledging the termination of their connection.

6.6 Sending Messages

Messages are sent by the client to a specific room. A message is constructed within the client's room chat bubble of the GUI. Once the user hits send, a the send message packet request [OP_CODE_SEND_MESSAGE(0x0000000C)] is created with the room name, client's username, and the message being sent. Upon receiving the send message request, the server then forwards the packet to all the users in that room. Each client will receive the message and display the contained message. Lastly, once all the clients in the room have received the forwarded message, another response packet is sent to the original sender confirming that their message has

been sent to the room. The opcode for this packet is [OP_CODE_SEND_MESSAGE_RESPONSE(0x0000000D)].

7. Server Messages/Responses

7.1 Listing Response

In the IRC chat there are two listing responses that the server takes care of Listing Rooms and Listing Users in a Room. Both of these responses return packets with either the names of the users or rooms depending on the request type. The response packet for listing all the active rooms contains the [OP_CODE_LIST_ROOMS_RESPONSE(0x00000005] opcode. Similarly, the opcode for the listing users response contains the [OP_CODE_LIST_USERS_RESP(0x00000007)] opcode. Once the response packet is received by the client, it is then its job to display this information to the user.

7.2 Forwarding Messages to Clients

Once the request packet[OP_CODE_SEND_MESSAGE(0x0000000C)] to send a message to a room is received by the server, this method is forwarded to all users in the room. It is then up to each individual client to display this information to the user. Encapsulated in the send message packet is the username of the sender, room name, and message being sent. Once this process is finished, the server sends a response packet[OP_CODE_SEND_MESSAGE_RESPONSE(0x0000000D)] to the original sender as confirmation of the message being sent.

8. Error Handling

Every time the server sends a request to the user, it waits for a response back from the server. In all cases it will receive one unless the server crashes or notifies all clients of its shut down beforehand. When the client receives the response packet, it first checks the opcode of the packet to see if an error occurred. There is a specific opcode indicating an error has occurred OP_CODE_ERR(0x00000001) which is what the client first reads to determine this. If the opcode is an error opcode, this indicates that the packet is an error packet. With all error packets there is specific error code associated with them (refer to section 3.2 of this RFC document for a list of all error codes). This is the next thing that is checked. Depending on the error code

returned and the type of request from the server, the client will handle the error and display an error message to the user when appropriate.

Both the client and the server must implement the exchange of keep alive messages in the case of an error on either side. Such an error could be due to the result of a crash or another unexpected circumstance. The exchange of these packets is carried out every five seconds during program execution until termination. The server is responsible for initiating and sending keep alive messages to all clients after the specified time interval and all clients are responsible for listening for these keep alive packets and responding with a subsequent packet.

9. Conclusion & Future Work

For future versions of the IRC chat client, we foresee being able to have one root menu and switchable tabs for the GUI in order to minimize the number of windows needed to use the IRC. Most importantly however, our next step would be to encrypt the messages being sent from the server to client and vise versa—this would be simplified with a cloud deployment. As developers we should not expect for the clients to encrypt their own messages. This lack of security allows for many security bugs making the chat very prone to attacks. We also hope to add the ability to send files, images and gifs through the chat with later versions.